HIGH LEVEL PETRI NETS WITH DATA STRUCTURE

C. Sibertin-Blanc
Université Toulouse 1 – Capitole
F-31042 Toulouse Cedex, France
sibertin@irit.fr

Abstract. For the lack of a data structure, Petri nets are not suitable for modeling systems such as Information Systems, where data have an effect on the system's behavior. In the High Level Petri Nets we set out here, tokens are replaced by entities which are described by means of a model closely related to those of the Database Theory. In order to take into account the values of entities, a *precondition* and an *action* may be associated to each transition. This data structure allows to define new nets' properties, which are analyzed by Program Theory techniques, whereas a lot of results provided by linear algebra remain valid. In addition, computation of invariants may be done.

The matter of this paper comes from the doctoral dissertation of the author.

6th European Workshop on Application and Theory of Petri Nets, K. Jensen Ed, Espoo, June 1985.

Petri Nets with Data Structure are High Level Petri Nets which aim at modeling Information Systems. So our main end isn't to describe system behaviour in a more concise way like Coloured (JEN 82), Predicate/Transition (GEN 81) or with Individual Tockens Nets (REI 83), but rather to take system data structures into account.

The behaviour of a system is related to its data structure in two ways. First elementary changes of state (which are described by transitions) more often are operations applying to system objects, and losing the object nature implies losing a part of operation meaning. Moreover, the system behaviour depends on the objects it processes ; for instance, book-keeping supplies differs from book-keeping a equipment, and the goods value decides of the holdig case.

The idea is to use a Data Model similar to Database Theory models in order to describe objects we call Entities, so that the marks are no longer tokens but Entities; moreover we associate to each transition a Precondition testing the Entity values and an Action processing them.

The linear algebra has proved to be a suitable framework for analysing Petri nets properties. These techniques remain very valuable for High Level Petri Nets analysis by using more sophisticated algebra than the integer one; the grounds of this approach are that High Level Petri Nets are foldings of ordinary Petri Nets. But proceeding only in this way becomes inadequate with High Level Nets as soon as they have an intrinsically higher expressive power and thus can't be unfolded as Petri Nets; the nature of the marks introduces new properties irreductible to algebraic analysis, and firing a transition sequence has certain similarities to program execution.

This paper introduces two such properties, ubiquity and coherence, and they are studied with techniques coming from Database and Program Theory. Allthough defined in the framework of our Petri Nets with Data Structure we believe they are common properties relevant for all High Level Petri Nets.

The language used to define and process the Net Data Structures is set out in the first section. In the second one, we recall some mathematical background we use in the following to define Petri Nets with Data Structure (PNDS). Sections 4 and 5 are devoted to ubiquity and coherence, and the last one to invariant calculus.

Since PNDS are a general modeling tool, we don't refer in this paper to Information System ; only an brief overview of Information System modeling with PNDS is given as a conclusion.

1) The data model

The model used to define and process the marks of PNDS is a conceptual one; it has a large expressive power, although it rests on three basic concepts only : Property, Entity Class and Entity.

A Property enables to describe an attribute, a piece of information, an item, ... of an object. It is defined by its name and a range of value ; this range of value is either a set of predefined constants (integers, boolean values, strings of characters,...), or an Entity Class. The value of a Property is an element or a subset of its range of value.

An Entity Class is a set of Properties, with a default value associated to each Property.

An Entity is an instance of its Entity Class : it is defined by its name, its Entity Class, and values tied up to the Properties of its Class ; Entities are the elements of PNDS markings.
When we need to distinguish between the name and the value of an Entity e, we denote them by e_n and e_v respectively. The value of properties are pointed out by the doted notation : if an Entity e has Property p, the expression e.p refers to the value of p in e.

In order to compute values, functions defined on the range of value of Properties may be used ; they are standard operations like arithmetical ones, or specific functions which are explicitly defined. These functions appear in terms, which are so defined expressions :

i) Entity names and Property values are terms ;
ii) Variables (with an associated range of value) are terms;
iii) If t is a term designating an Entity and p is a Property of this Entity, then t.p is a term;
iv) If t_1, .., t_r are terms and f is a computable function, then $f(t_1, .., t_r)$ is a term.

This Data Model stems from the Minsky frames (MIN75). It enables to describe the data structure of Information Systems, and it has the same expressive power as models used for Data Processing or Office Automation ; concerning the main models of the Database Theory (entity-relationship, relationnal, network and hierarchical), there is an unambiguous mapping from their concepts to those of this Data Model (SIB84).

Moreover, this Data Model brings two advantages.
First, it is a conceptual model without sophisticated mechanisms; this enables to bring to the fore the joint between the data structure and the working of Systems, which is expressed in a simple way by means of a few basic concepts.
The second advantage is of a practical nature ; when the whole data structure of a System is described with any particular model, the part which interacts with the working may be translated into the data model of PNDS, so much so that PNDS may still be used to describe the system behaviour ; thus modeling with PNDS is not dependent on a given data model.

2) Mathematical notations

We will give a few mathematical notations needed for the Petri nets using marks which are tuples of individuals.

Let E be a set; we note Z(E) the commutative free group generated by E ; with additive notation, Z(E) is the free Z-module generated by E too; his elements are called multisets and expressed with the following notation :

$x = \Sigma_{e \in E}\ x(e)\ e$ where $x(e) \in Z$ and only finitely many $x(e)$ are not null.

Z(E) is partially ordered by the so defined relation :
$x < y$ iff $x(e) < y(e)$ for each e in E.
We denote by N(E) the set of positive multisets.

The size of a multiset is the sum of its coefficients; it defines a linear function from Z(E) to Z, denote by !.! :

$$|x| = \Sigma_{e \in E}\ x(e).$$

Let i be the canonical embedding from E to Z(E) :

$$i : E \longrightarrow Z(E)$$
$$e \longmapsto 1\ e$$

We will consider E as a subset of Z(E) (instead of i(E)) and so i(e) will be denoted by e.

Let E and F be two sets, and f a function from E to F. The group extension of f is the function f' :

$$f' : Z(E) \longrightarrow Z(F)$$

such that : $f'(0) = 0$,

$$f'(x) = \Sigma_{e \in E}\ x(e) f(e).$$

f' is the group homomorphism which extends f, and it is surjective or injective if and only if f does. We usually don't distinguish f' from f, without damage.

We denote E^* the free monoide gerated by E ; its elements are tuples of elements of E, of the form :

$x = \langle e_1, .., e_n \rangle$, where $e_i \in E$ for $i = 1 .. n$.

The neutral element of E^*, whose length is null, is denoted by 1.

Let E and F be two sets, and f a function from E to F. The monoide extension of f is the function :

$$f^* : E^* \longrightarrow F^*$$

defined by : $f(1) = 1$,

$$f(\langle e_1, .., e_n \rangle) = \langle f(e_1), .., f(e_n) \rangle.$$

f^* is the monoid homomorphism wich extends f, and they can be mixed up.

We will have to consider multisets of tuples, and introduce two more definitions devoted to them.

We define the degree of an element e of E in a multiset x of $N(E^*)$ as the occurence number of e in x. This function is denoted by $d_x(e)$ and is defined by linearity from the value it takes on E :

i) $d_0(e) = 0$

ii) $d_{\langle e' \rangle}(e) = 1$ if $e' = e$
$= 0$ else ;

iii) if x is of the kind $f^*.g^*$, where f^* and g^* are elements of E^*, then $d_x(e) = d_{f^*}(e) + d_{g^*}(e)$;
iv) if x is of the kind f+g, where f and g are elements of $N(E^*)$, then $d_x(e) = d_f(e) + d_g(e)$.

For example, if $z = 2 + 2\langle e_1, e_2, e_1\rangle + \langle e_2, e_3\rangle$, then $d_x(e_1) = 4$, $d_x(e_2) = 3$, $d_x(e_3) = 1$ and $d_x(e_4) = 0$.

The degree is a linear function with respect to x, and the following equality holds :

$$d_x(e) = \sum_{e^* \in E^*} x(e^*).d_{e^*}(e).$$

Finely, we define the support of a multiset x of $N(E^*)$ as the set of elements which appear in it :

$$supp(x) = (e \in E \ ; \ d_x(e) \geq 1).$$

We have $supp(x+y) = supp(x) \cup supp(y)$, but $supp(x-y) = supp(x) - supp(y)$ holds if and only if either $d_y(e) = 0$ or $d_y(e) = d_x(e)$ for each e of E ; (for instance it holds if x and y are sets).

3) Petri Nets with Data Structure

Definition 1:

A Petri Net with Data Structure is defined by the following items :

1) A set P of places, for which we denote card (P) = m.
2) A set T of transitions, for which we denote card (T) = n.
3) A set EC of Entity Class, which are the Class of Entities of the net markings.
4) A set V of variables.
5) A function denoted by cl which defines the Class of places and variables :

cl : P -----> EC^*
cl : V -----> EC which is extended from $Z(V^*)$ to $Z(EC^*)$.

The class of a place determines the Entity Class of Entity tuples it may receive (a place whose class is 1 is called a token place). The class of a variable determines its range, i.e. the Entity Class of Entities which may be substituted for it.

6) A forward incidence function :
$Pre : TxP \longrightarrow N(V^*)$
such that there exists an integer k :
$cl(Pre(t, p)) = k.cl(p)$
(the tuples of Pre(t, p) have to have the same class as p).
If $Pre(t,p) \neq 0$, p is called an incoming place of t, and t an outcoming transition of p.
$V_i(t)$ denotes the support of $\Sigma_{p \in P} Pre(t,p)$, whose elements are the incoming variables of t.

7) A backward incidence function :
$Post : T \times P \longrightarrow N(V^*)$
for which the same constraint and definitions hold.

8) A Precondition and an Action are associated to each transition, and they will be defined below.

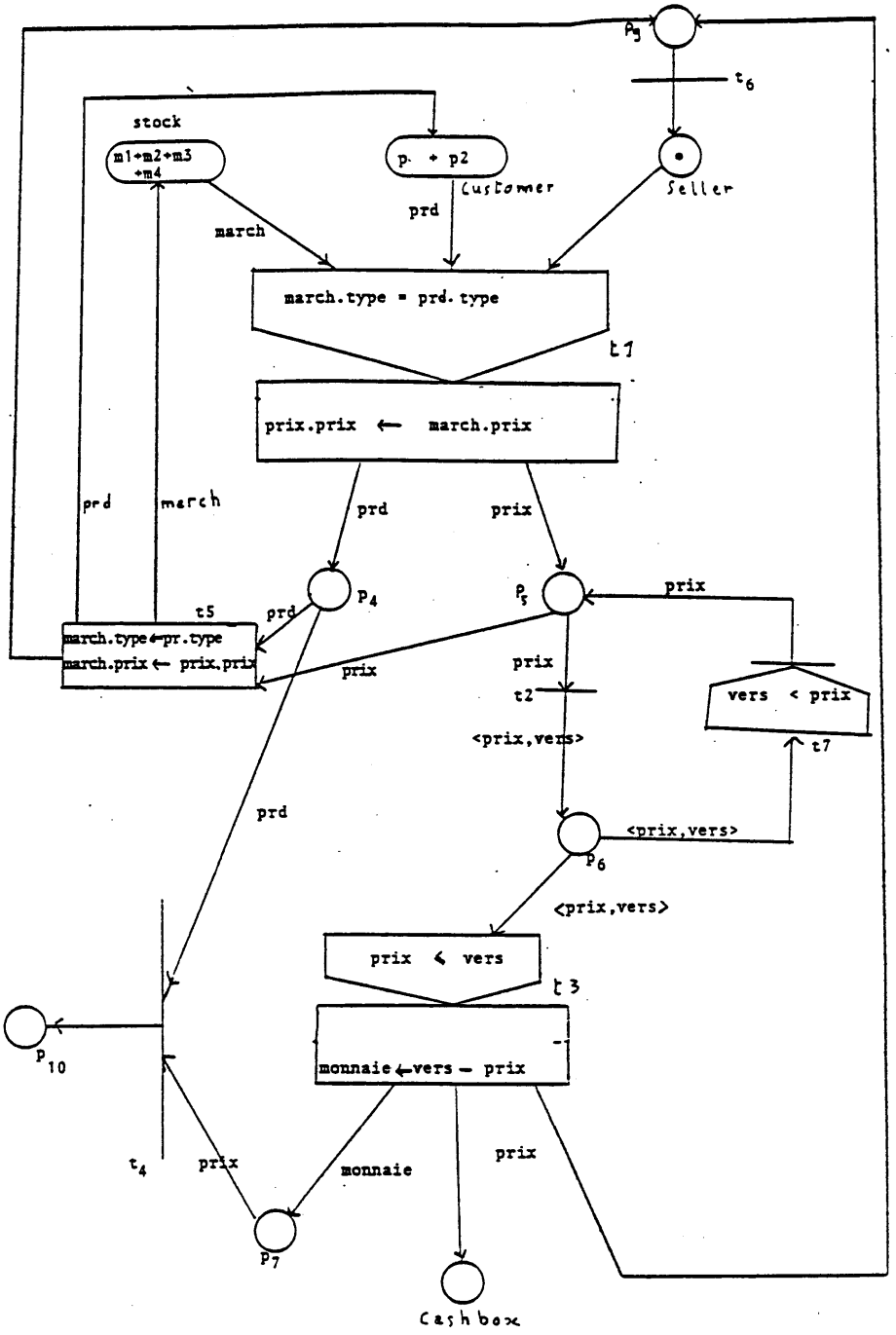
Définition 2:
A marking of a PNDS is a couple $M = (M_E, M_d)$ where :
1) M_E is a set of Entities whose Class belong to EC, so that we can define :
$cl : M_E \longrightarrow EC$
2) M_d is a distributing function which determines the marking of each place :
$M_d : P \longrightarrow N(M_E^*)$
such that there exists an integer k, $cl(M_d(p)) = k.cl(p)$
(the tuples of Entities of $M_d(p)$ have to have the same class as p).

To be more rigorous, M_d takes its values in $N(M_{E_n}^*)$, because the distributing of Entities only refers to their name, not to their value.
The neutral element of M_E^* is called token.

We will give a simple example in order to illustrate these definitions.

EXAMPLE

The matter is to model the behaviour of a seller and a customer at a counter. When they are together, the customer asks for some goods ; the seller gets the merchandise in its stock and notifies the customer of the price ; if the price isn't right then the seller puts the merchandise away ; else the customer pays the price and takes the merchandise, while the seller collects the payment and gives the change back.


stock
m1+m2+m3 +m4
p1 + p2
Customer
Seller
prd
march
march.type = prd.type
t1
prix.prix ← march.prix
prd
march
prd
prix
P4
P5
prix
t5
prd
march.type←pr.type
march.prix ← prix.prix
prix
prix
t2
vers < prix
t7
<prix,vers>
prd
<prix,vers>
P6
<prix,vers>
prix ≤ vers
t3
P10
monnaie ←vers − prix
t4
prix
monnaie
prix
P7
Cash box

The structure of the corresponding Net is represented on figure 1, where Preconditions are inscribed in pentagones before transitions, and Actions in rectangles. The figure does neither show the set EC of Entity Class nor the cl function; they have to be defined separately.

The Entity Class are the three following ones :

```
Class Merchandise :
     nature : (apple . . yogurt);
     price : real dec(2).

Class Goods :
     nature : (apple . . yogurt).

Class Payment :
     amount : real dec(2).
```

The Class of the variables is so defined :

```
     price ,payment, change : Payment ;
     merchandise : Merchandise ;
     goods : Goods.
```

The Class of places is infered from the class of variables occurring in the incidence functions.

Figure 1 represents the distributing function M_d of a marking, but the value of Entities has to be defined separately too.

For instance we may have :

```
Merchandise m1 :                    Merchandise m2 :
     nature = apricot;                   nature = apricot;
     price = 12.40.                      price = 15.

Merchandise m3 :                    Merchandise m4 :
     nature = pear;                      nature = cherry;
     price = 6.                          price = 10.

Goods g1 :                          Goods g2 :
     nature = apricot.                   nature = pear.
```

***

The behaviour of PNDS is in accordance to the common rules governing High Level Petri Nets : a transition is loaded if its incoming places contain enough Entities so that the variables of the incidence function Pre may catch different Entities; moreover, the caught Entities have to satisfy the Precondition of the transition. Firing a transition will remove Entities from the incoming places, alter their value according to the Action, and put Entities in the outcoming places.

Definition 3 :

Let $M = (M_E, M_d)$ be a marking of a PNDS and t a transition.
A substitution is a function :

$S: V_i(t) \longrightarrow M_E$ such that $cl(S(v)) = cl(v)$ and which is extended to $Z(V_i(t)^*)$

(In fact, substitutions take their values in set of Entity names, as distributing function do).

A transition t is said to be loaded from M iff there exists a substitution S such that :

$S(Pre(t,p)) \leq M_d(p)$, for each place p.

This is denoted by $M((t,S)$.

The loading of transitions is an algebraic condition unrelated to the data structure which doesn't take into account the entity values ; this is the part of Preconditions.

A Precondition is a boolean expression (without quantifier) where variables are incoming variables of the transition and terms are as defined in section 1.

Definition 4:

Let (t, S) be a transition loaded from M and $Pr(v_1,..,v_r)$ the Precondition of t. (t,S) is firable if and only if the proposition $Pr(S(v_1),..,S(v_r))$ is true.

This is denoted by $M((t,S)>$.

EXAMPLE

The transition t_1 of figure 1 example is loaded by the eight substitutions :

$S_{i,j}$: merch :--> m_i i=1,..,4
goods :--> g_j j=1, 2.

But they are only three firable loaded transitions : $(t_1, S_{1,1})$, $(t_1, S_{2,1})$ and $(t_1, S_{3,2})$, according to the above given values.

Transitions t_3 and t_7 are always simultaneously loaded ; but they are not simultaneously firable because they have contradictory Preconditions.
***

When a firable from $M = (M_E, M_d)$ transition (t,S) is fired, a new marking $M' = (M'_E, M'_d)$ is reached.

The new distributing function is defined according to the incidence functions and the substitution ; for each outcoming variable which is not an incoming one, a new outside M_E Entity is generated, so that S is extended to $V_i(t) \cup V_o(t)$; then for each place p,

$$M'_d(p) = M_d(p) - S(Pre(t,p)) + S(Post(t,p)) \qquad (I)$$

The set of Entity names of M_E is defined as the support of $\sum_{p \in P} M'_d(p)$; usually we don't have :

$$M_{E_n}' = (M_{E_n} - S(V_i(t))) \cup S(V_o(t)) \qquad (II)$$

because $\sum_{p \in P} Pre(t,p)$ and $\sum_{p \in P} Post(t,p)$ are not sets but multisets.

The value of M_E' Entities is defined according to the transition Action and the substitution.

An Action is a set of assignments of one of the following kinds :

$v \leftarrow f(v_1,..,v_r)$
$v.p \leftarrow f(v_1,..,v_r)$

where v is an outcoming variable of t,
p is a Property of the Entity Class cl(v),
f is a term whose variables are incoming ones.

So the Value of M_E' Entities results executing statements such as :

$S(v) \leftarrow f(S(v_1), .., S(v_r))$
$S(v).p \leftarrow f(S(v_1), .., S(v_r))$

provided by substituting Entities for variables in the t Action. (The statements are supposed to be executed in parallel).

An interpreter of PNDS has been build up (SIB 84), using the Object Programing language MERING (FER 83).

Two mechanisms related to the Entity catching may be added to this definition of PNDS.

The first one is to associate Registers to a Net ; Registers are steady Entities which belong to all marking of the Net but aren't located in places ; they are global Entities that Precondition and Action of any transition may refer to.

The second mechanism is to authorize having Entity names in the set of variables ; such a variable may only catch the same named Entity.

It is easy to verify these extensions don't extend the expressive power of PNDS.

Now it is worthwhile to compare PNDS to other High Level Petri Nets such as Predicate/Transition Nets (GEN 81), IML-inscribed Nets (RICH 81) or Object-Actor Nets (OBER 82).

PNDS seem to be defined in a more formal and explicit way ; for instance, variables are typed and calculation of outcoming marks is set apart from their distributing (since Action and backward incidence function are not mixed up). So PNDS analysis is easier, like their interpretation.

The challenge of HLPN is on one hand to have a tool for modeling accurately complex systems, including their data structure, and on the other hand to be able to analyse system properties, even those involving their data structure.
Concerning Pr/T Nets, the marks are handled as constants (as long as the structure is not defined) and the resulting expressive power is too low.
Concerning Object-Actor and IML-inscribed Nets, they use a Data Definition Language which is very suitable for modeling but not convenient for mathematical treatements. The same holds for Augmented Petri Nets (ZIS 75), which in addition have recourse to production rule systems.

In order to take advantage of Petri Net Theory results, we associate an ordinary Petri Net to a PNDS.

Definition 5 :

Let R = <P, T, EC, V, cl, Pre, Post> be a PNDS. The underlying net of R is the Petri Net :
$\underline{R}$ = <P, T, $\underline{Pre}$, $\underline{Post}$> such that for each place p and transition t :

$\underline{Pre}(t,p) = |Pre(t,p)|$,
$\underline{Post}(t,p) = |Post(t,p)|$.

If M is a marking of R, its underlying marking is the marking $\underline{M}$ of $\underline{R}$ defined by

$\underline{M}(p) = |M_d(p)|$ for each place p.

So the underlying net of a PNDS is obtained by becoming unaware of its data structure ; Entity and variable tuples are turned into tokens, but places, transitions and their links remain unaltered.

It is valuable to relate properties of a PNDS to those of its underlying net, since they are easier to analyse. The two following sections of this paper are mainly devoted to relationships between PNDS and underlying nets properties.

A PNDS is bounded if and only if its underlying net does.

Concerning net behaviour, definitions 3, 4 and 5 lead straight to :

Theorem 1 :
Let (R, M) be a marked PNDS, ($\underline{R}$, $\underline{M}$) its underlying net and (s, S) a loaded sequence firable from M. Then s is firable from $\underline{M}$ in $\underline{R}$.

We are interrested in converses of this theorem.
An exact converse would mean the net data structure has no effect on its behaviour and is worthless : such a net is equivalent to an ordinary Petri net. Then we look for cleverer converses and to this end introduce two new properties : ubiquity and coherence.

4) Ubiquity

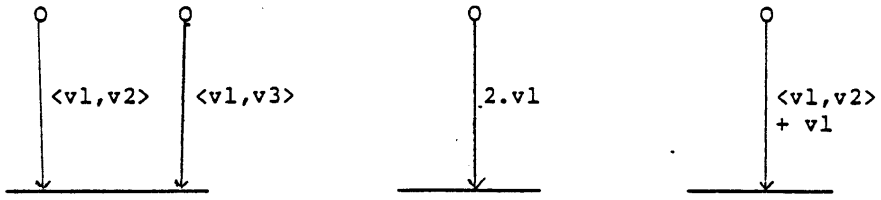
The first property we study is of algebraic matter and is relevant for every High Level Petri Net with individual marks. It concerns the mark ability to have several occurences in a marking and has valuable consequences.

Definition 6 :
A PNDS is without ubiquity iff for each transition t and variable v :

$$d_{\Sigma_{pe}Pre(t,p)}(v) \leq 1,$$
$$d_{\Sigma_{pe}Post(t,p)}(v) \leq 1$$

A marking is without ubiquity iff for each Entity e,
$d_{\Sigma_{p \in P} M_d(p)}(e) = 1$.

(In the following sums are on the set P of places).
For instance, there is ubiquity in the following cases :



To make a PNDS without ubiquity is always possible by altering either its data structure, or the structure of its underlying net, or both.

A without ubiquity marked Net remains such one:

Proposition 1
Let (R, M) be a without ubiquity marked PNDS, and M' a reachable from M marking. Then (R, M') is without ubiquity.

Proof
We have to prove M' is without ubiquity, and may suppose there exists (t,S) such as M ((t,S)> M'.

First let notice S is an injective substitution. Else, let v and v' be variables such that S(v) = S(v') = e. Then
$d_{\Sigma S(Pre(t,p))}\ S(v)) > 1$,

and as $S(Pre(t,p)) < M_d(p)$, for each place p,
$d_{\Sigma M_d(p)}(e) > 1$ and M is ubiquitous.

Since S is injective, for each multiset x of N(V*) and for each variable v we have :
$d_{S(x)}\ (S(v)) = d_x(v)$.

Moreover for each place p we have :
$M'_d(p) = M_d(p) - S(Pre(t,p)) + S(Post(t,p))$.

Thus for an Entity e such as S(v) = e, the linearity of d leads to :

$$d_{\Sigma M'_d(p)}(e) = d_{\Sigma M_d(p)}(e) - d_{\Sigma Pre(t,p)}(v) + d_{\Sigma Post(t,p)}(v).$$

The hypothesis is that the right member terms are bounded by 1, and we ask for the left member to be 1. If v is a not incoming outcoming variable, it holds because the first terms is null ; else it holds too because the second term is 1.
***

PNDS without ubiquity are of an easier use for two reasons.

The multiset equipment becomes superfluous defining PNDS without ubiquity, and only the more familiar sets are needed ; this leads to several simplifications such as the equation (II) of section 3, and system modeling with such Nets requires less mathematical knowledges.

Moreover, non-ubiquity avoids the data sharing problem. A given Entity can't be simultaneously caught by several loaded transitions so that Entity updating remains safe even in case of parallel firing . Thus a number of transitions may be fired at a step and we have a kind of persistency :

Proposition 2 :
Let (R,M) be a pure marked PNDS, (t_1,S_1) and (t_2,S_2) firable from M transitions, and M' the marking such as $M\ ((t_1,S_1) > M'$. Thus (t_2,S_2) is firable from M' if it is loaded.

We now state the converse of theorem 1 related to non-ubiquity.

Theorem 2 :
Let (R,M) be a without ubiquity marked PNDS, $(\underline{R},\underline{M})$ its underlying net, and s a transition sequence such as $\underline{M}\ (s>$ in $\underline{R}$.
Then there exists a substitution sequence S such as (s,S) is loaded from M in R.

Proof
We may suppose s = t by induction on the s length, and will build a substitution S such that $M\ ((t,S)$.

Let p be an incoming place of t ; since R is without ubiquity, noting a_p the arity of p we have :

$$Pre(t,p) = \langle v_{1,1},..,\ v_{1,a_p}\rangle + .. + \langle v_{r,1},..,\ v_{r,a_p}\rangle$$
where $v_{i,j} \neq v_{h,k}$ if $(i,j) \neq (h,k)$.

Since $\underline{M}$ (t >, we also have :
$M_d(p) = \langle e_{1,1},\ldots, e_{1,a_p}\rangle + \ldots + \langle e_{u,1},\ldots, e_{u,a_p}\rangle$ where $r \leq u$.

Then we may define a partial substitution :
S_p : supp(Pre(t,p)) ---> M_E
$v_{i,j}$:---> $e_{i,j}$

Now we have, for $p \neq p'$:
supp(Pre(t,p)) $\cap$ supp (Pre(t,p')) = $\emptyset$,
so we may define :
S : $V_i(t)$ ----> M_E
$v_{i,j}$:---> $S_p(v_{i,j})$ where $v_{i,j} \in$ supp(Pre(t,p)), and $S(Pre(t,p)) \leq M_d(p)$ holds for each p.

***

We don't have equivalence in this theorem, as in proposition 1. It does not means non-ubiquity is too strong a condition : without ubiquity PNDS are exactly the ones for which both hold.

5) Coherence

We believe the coherence property we introduce in this section is the fundamental one of PNDS, like the bounded and live properties are for ordinary Petri Nets.

It is defined as the theorem 1 converse holding, and so enables to keep as properties of an PNDS the properties of its underlying net ; concerning modeling power, coherence informally means :

> Let R be a PNDS describing a system S.
> Then R is coherent iff its underlying net $\underline{R}$ describes S too, at a lower detail level.

Definition 7 :

> Let R be a RPDS, m a distributing function of R and $(\underline{R},\underline{m})$ the underlying net of (R,m). (R,m) is coherent iff for each sequence $s = t_1..t_r$ of transitions such that $\underline{m}$ (s> in $\underline{R}$, there exists a marking M = (E,m) of R from which s may be fired, i.e there exists a set of entities E and substitutions S_i (i=1..r) such that :

$m : P \longrightarrow N(E_n^*)$
$S_i : V \longrightarrow E_n$
$M ((t_1,S_1) \ldots (t_r,S_r) >.$
R is said to be coherent if it is coherent for every distributing function m.

Let notice that a ubiquity-less PNDS which has empty Preconditions is coherent, according to theorem 2.

So far, the living property has not be defined for High Level Petri Nets. The following definition seems to be the most suitable :

Definition 8 :
Let R be a RPDS and m a distributing function of R. (R,m) is live iff :
i) its underlying net ($\underline{R}$,$\underline{m}$) is live ;
ii) (R,m) is coherent.

Since coherence is a very important property, we have to study its decidability.

Coherence is relevant for all High Level Petri Net, like ubiquity, but it isn't of algebraic matter.It relies straight upon the data structure and its study requires to view PNDS as a parallel programming language.

Definition 9 :
Let $(s,S) = (t_1,S_1) \ldots (t_r,S_r)$ be a (finite) loaded sequence of a PNDS, where the S_i take their values in the Entity set E, and denote the Precondition and Action of t_i by Pr_i and A_i.

i) the program of (s, S) is the predicate and assigment sequence :
$P(s,S) = S_1(Pr_1), S_1(A_1)$;
.
.
.
$S_r(Pr_r), S_r(A_r)$.

ii) the outcoming of P(s,S) execution from E at step i, denoted by P(s,S) (i,E), is so defined by induction :
if i=1, P(s,S) (1,E) results of applying $S(A_1)$ to E;
$P(s,S) (i,E) = P(t_i,S_i) (1, P(s,S) (i-1,E))$;
we note $P(s,S) (r,E) = P(s,S)(E)$.

iii) we say P(s,S) may be executed from E at step i iff :
if i=1, E satisfies the predicate $S_1(Pr_1)$;
if i > 1, P(s,S) may be executed from E at step i-1 and P(s,S) (i-1,E) satisfies $S_i(Pr_i)$.
P(s,S) may be executed from E if it does at step r.

With this equipment, we can characterize firable sequences in PNDS.

Proposition 3 :
Let :
R a PNDS, M = (E,m) a marking of R,
$s = t_1 \ldots t_r$ a sequence of transitions,
$S = S_1 \ldots S_r$ a sequence of substitutions.

Thus, there exists a marking $M_1 = (E_1,m_1)$ such that $M((s,S) > M_1$
if and only if :
i) for $i = 1, \ldots, r$, t_i is loaded by S_i from $m + S_1(C.\underline{t}_1) + \ldots + S_{i-1}(C.\underline{t}_{i-1})$;
ii) P(s,S) may be executed from E.

Moreover, $m_1 = m + S_1(C.\underline{t}_1) + \ldots + S_r(C.\underline{t}_r)$ and $E_1 = P(s,S)(E)$.

Proof
By induction on the length of s, the proposition arises straightforward from definitions 3, 4 and 9.
***

If R and m are without ubiquity, theorem 2 enables to replace the i) condition of this proposition by the following one :
i') $\underline{m}$ (s > in the underlying net $\underline{R}$.

Then the property to be firable may be divided into two unrelated properties, the first one dependent on the underlying net and the second one on the data structure of the net. We believe this result, like definition 8, is a meaningful example of the way to analyse PNDS, by breaking down their properties into, on the one hand general properties of ordinary Petri Nets, and on the other hand properties concerning their data structure.

Now, we will show that the condition ii) of proposition 3 is decidable.

Proposition 4 :

Let :
R a PNDS, m a distributing function of R,
$s = t_1 .. t_r$ a finite sequence of transitions,
$S = S_1 .. S_r$ a sequence of substitutions such that
for $i = 1 .. r$, t_i is loaded by S_i
from $m + S_1(C.\underline{t_1}) + . . + S_{i-1}(C.\underline{t_{i-1}})$.
Thus, the existence of a set E of Entities such that P(s,S) may be executed from E is decidable.

Proof

The idea is to bring all the Preconditions back to the beginning of the program, in such a way as the program may be executed from a set of Entities E if and only if E satisfies the initial Precondition (HOARE69). The existence of such a set E is thus equivalent to the consistency of this Precondition, which is kown to be decidable.

By induction on the length of s, we may suppose r = 2.

Let $EN = (e^1,...,e^r)$ be a set of Entity names in which the distributing function m and the substitutions S_i take their value, and denote $e^* = (e^1,...,e^r)$.

By reordering variables in Preconditions and Actions, we have : $P(s,S) = Pr_1(e^*), A_1 ;$
$Pr_2(e^*), A_2 .$
where A_i is of the kind :
$e^* <-- f_i(e^*)$.

Let now E be a set of Entities whose names are in EN, i.e. $E_n = EN$. According to definition 9, we have the following equivalences :

P(s,S) may be executed from E,
<==> P(s,S) may be executed at first step from E
and $P(s,S)\ (1,E)$ satisfies $Pr_2(e^*)$,
<==> P(s,S) may be executed at first step from E and E satisfies $Pr_2(f_1(e^*))$,
<==> P' may be executed (at first step) from E,

where P' is the following program :
$P' = Pr_1(e^*)$ and $Pr_2(f_1(e^*)), A_1 ; A_2$.

When Actions are not made up of simple terms but of functions of the kind If Then Else, we may process in the same way. Indeed, let A_1 be of the kind :

e^* <-- IF $C(e^*)$ THEN $f_{1,1}(e^*)$ ELSE $f_{1,0}(e^*)$.

Then, the initial Precondition of P' becomes :

$Pr_1(e^*)$ and (($C(e^*)$ and $Pr_2(f_{1,1}(e^*))$)
or (not $C(e^*)$ and $Pr_2(f_{1,0}(e^*))$)).

***

Theorem 3

Let R be a PNDS, m a distributing function of R and ($\underline{R}$,m) the underlying net of (R,m).
If the set of transition sequences s such that $\underline{m}$ (s> in $\underline{R}$ is finite, then the coherence of (R,m) is decidable.

Proof

We will reduce the coherence of (R,m) to the consistency of a Precondition ; this Precondition is the conjunction, for all sequences s fireble from $\underline{m}$ in $\underline{R}$, of disjunctions of initial Preconditions of programs, for all loaded sequences (s,S) from m in R.
Since the set of transition sequences s such that $\underline{m}$ (s > in $\underline{R}$ is finite, deciding the coherence of (R,m) is reduced to deciding the existence, for each s, of a substitution sequence S and of a set of Entities E such that (E,m) ((s,S) > in R.

Let s be any sequence of transitions such that $\underline{m}$ (s > in $\underline{R}$; the length of s is finite, since there are only finitely many sequences fireble from $\underline{m}$ in $\underline{R}$.

First we will show that the set of substitution sequences S such as (s,S) is a loaded sequence from m, is also finite and may be effectively computed.
Indeed, let EN be the range of value of the distributing function m, and t the first transition of S. Since EN and the variable set V are both finite the substitutions

$S_1 : V \dashrightarrow EN$

such that (t,S_1) is loaded from m in R are effectively computable and finitely many. Since s is of finite length, repeating this process produces the (finite) set L(s) of all sequences S such that (s,S) is loaded from m.

Let now $Pr_{s,S}$ be the initial Preconditions obtained from P(s,S) like in Proposition 4, and Pr_s be the disjunction of $Pr_{s,S}$,, for S in L(s). Thus we have the following equivalences :

Pr_s is consistent,
<==> there exists S such that $Pr_{s,S}$ consistent,
<==> there exists S and E such that M((s,S)>, where M=(E,m) is a marking of R.

Thus, defining Pr as the conjunction of Pr_s, for s such that $\underline{m}$ (s > in $\underline{R}$, (R,m) is coherent iff Pr is consistent. In order to decide the consistency of Pr it remains to build the set of sequences s such that $\underline{m}$ (s > in $\underline{R}$; but ($\underline{R}$, $\underline{m}$) is bounded since this set is finite, and it can be computed from the marking tree of ($\underline{R}$, $\underline{m}$).
***

The scope of this theorem is limited because of its strong hypothesis, but we believe it may be improved.

In fact, it states that coherence is decidable if we only consider the sequences of transitions whose length is finite. To be more precise, let define the h-coherence as the following property : for each sequence s whose length is less than h and such as $\underline{m}$ (s > in $\underline{R}$, there exist S and M such that M ((s,S)' in R and $\underline{M}$ = $\underline{m}$. Thus the h-coherence is decidable according the above proof.

6) Invariant calculus

An invariant of a net is a statment about its behaviour, and it is necessary to compute invariants in order to make validations when system modelling. So we will define PNDS invariants and indicate how to compute them.

PNDS have a large variety of invariants which either are of algebraic matter or rely upon the data structure. A method of systematic computation will be given for algebraic invariants ; but data invariants are much more difficult to formalize and we will only give an example.

Let us recall some notations of matrix calculus.
Let A and B be Z-modules,
$W = (W_i)_{i\in I}$ a vector of A,
$V = (V_i)_{i\in I}$ a vector of integers,
$C = (C_{i,j})$ a matrix of A,
$F = (F_i)_{i\in I}$ a vector of functions from A to B.
Then we denote :
$V.W = \Sigma_{i\in I} V_i.W_i$ where the dot in the right member is the scalar product in A;
$V.C = (\Sigma_{i\in I} V_i.C_{i,j})_j$;
$F.W = \Sigma_{i\in I} F_i(W_i)$;
$F.C = \Sigma_{i\in I} F_i(C_{i,j}))_j$.

According to (JEN82) and (VAUT84), PNDS invariants are so defined :

Definition 10 :

Let EN be Entity name set of markings of a PNDS R, and B a Z-module.
A weighting of R is a vector $W = (W_p)_{p \in P}$ of functions :

$W_p : Z(EN^*) \longrightarrow B$

A weighting W is an invariant iff for every marking M and M' such as $M (>M'$, $W.M_d = W.M'_d$ holds.
W is linear iff the W_p are so.

For all kinds of Petri Nets, invariant studying is based on a Lautenbach's Theorem (LAUT74) ; its PNDS version is :

Proposition 5

Let W be a linear weighting. W is an invariant iff for each substitution $S : V \longrightarrow EN$
we have $W.S(C) = O$, where C is the incidence matrix of the net.

Proof

Let W satisfy the condition and M and M' be two markings such as there exists (t,S), $M ((t,S)> M'$; thus W is an invariant since it is linear and $M_d' = M_d + S(C).\underline{t}$.

Conversely, let (t, S) be a loaded transition ; there exists M and M' such as $M ((t, S)> M'$, and thus $W.S(C).\underline{t} = 0$ if W is a linear invariant. This holds for each transition t, and thus $W.S(C) = 0$.

***

Definition 11 :

A linear weithting $W = (W_p)$ is called algebraic iff there exists a function F and a set X,

$F : EN^* \cup V^* \longrightarrow X$

such as :

i) for each substitution S, there exist an extension :

$S_F : V^* \cup X \longrightarrow EN^* \cup X$

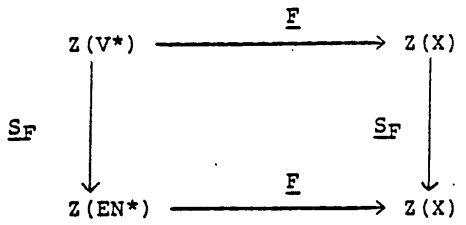
such as for each v^* in V^* :

$S_F (v^*) = S (v^*)$,

$F \circ S(v^*) = S_F \circ F(v^*)$

ii) $W_p = w_p.\underline{F}$, where w_p is an integer and $\underline{F}$ is the extension of F to $Z(EN^* \cup V^*)$.

Let denote $\underline{F}$ and $\underline{S_F}$ the linear extensions of F and S_F respectively ; thus the following diagram commutes :



Now we have :

Theorem 4 :
Let $W = (w_p.F)$ be an algebraic weighting.
W is an invariant iff $W.F(C) = 0$.

Proof
According to proposition 5, we have only to show :
$W.F(C) = 0$ iff $W.F(S(C)) = 0$ for each substitution S.

Let suppose $W.F(S(C))=0$ for each substitution, S be an injective substitution and $\underline{S}$ be the extension of S_F. $\underline{S}$ is injective too and linear, so it has a null kernel (i.e. $\underline{S}(m) = 0 \Rightarrow m = 0$).
Thus we have for each transition t :
$\Sigma_p w_p.F(S(C_{p,t})) = S(\Sigma_p w_p.F(C_{p,t})) = 0.$

Then $\Sigma_p w_p.F(C_{p,t}) = 0$ for each t, i.e. $W.F(C) = 0$.

The converse implication is obvious by linearity.
***

Such invariants are called algebraic because their computation involves only the incidence matrix of nets ; the value and even the structure of marks of net markings is not taken into account, so they apply to every High Level Petri Net.
Each function F as in definition 11 defines an invariant type, so there are many invariant types for PNDS. Some having a general meaning are relevant for each Net, and others are specific : their interpretation refers to the considered net and the system it models.

Now we will give some examples of general algebraic invariants. $W = (w_p)$ will denote a vector of |P| integers, and M and M' two any markings. The support of an invariant $W = (w_p.F)$ is the set of places p such as $w_p \neq 0$.

1. The identity

The identity function of $Z(EN^* \cup V^*)$ commute with every substitution, and thus defines an invariant type :

$(M (> M' \Longrightarrow w.M_d = w.M_{d'})$
iff $w.C = 0$.

If a Net has such invariants there exist permanent Entity tuples which always stay in places of the invariant support.

2. The degree

Let recall the degree $d_m(e)$ of a element e in a multiset m is the occurence number of e in m. So we define a linear function :

$D_E : N(E^*) \longrightarrow N(E \cup (1))$
$m \longmapsto \sum_{e \in E} d_m(e).e + m(1)$

which transforms products of E^* into sums of $N(E \cup (1))$. For instance, if $m = 3 + 2.\langle e1, e3\rangle + \langle e3, e4\rangle$ then
$D_E(m) = 3 + 2.e1 + 3.e3 + e4$.

Let consider $D = D_{E_n} \cup V$,
$D : N((E_n \cup V)^*) \longrightarrow N(E_n \cup V \cup (1))$
and verify it commutes with substitutions.
If S is any substitution and $v^* = \langle v_1, . ., v_r\rangle \in V^*$, we have :

$$\begin{aligned} D \circ S(v^*) &= D(\langle S(v_1), . ., S(v_r)\rangle) \\ &= S(v_1) + . . + S(v_r) \\ &= S(v_1 + . . + v_r) \\ &= S \circ D(v^*). \end{aligned}$$

Thus D defines an algebraic invariant type since it is linear, and

$(M (> M' \Longrightarrow w.D(M_d) = w.D(M'_d))$
iff $w.D(C) = 0$.

If a Net has such invariants, there exist permanent Entities and tokens which always stay in places of the invariant support, but not necessarily gathered in permanent tuples.

Tokens may be left by removing the term m(1) in D_E definition.

3. The class

Let cl be the function which defines the Entity Class of variables and Entities; we denote cl its linear extension too :

$cl : N((E_n \cup V)^*) \longrightarrow N(EC^*)$.

For each substitution S and variable v, we have :
$cl(S(v)) = cl(v)$ by definition.

Extending S on V U EC by S(c) = c for any Entity Class c, its linear extension commutes with cl; thus cl defines an algebraic invariant type, and we have :

(M (> M' ==> $w.cl(M_d) = w.cl(M'_d)$)
iff $w.cl(C) = 0$.

If a Net has such invariants, there exist permanent Entity Class tuples which always have instances in places of the invariant support.

Composing cl with the above defines function D yields a new invariant type concerning the permanence of Entity Class instances.

4. Projections.

Let E' be a subset of a set E; the projection of E onto E' is the monoide homomorphism :

$Pr_{E'} : E^* \longrightarrow E'^*$

defined by : $Pr_{E'}(\langle e \rangle) = \langle e \rangle$ if e is in E'
1 else.

These projections don't define invariants, because they don't commute with all substitutions. (It is the case only if each Entity is always caught by the same variable).

However, composing such a projection with the function cl yields an algebraic invariant type since we have :

Pr o cl o S = Pr o S o cl = Pr o cl = S o Pr o cl.

This invariant type concerns tuple instances of Entity Class not avoided by the projection.

There exists another kind of projection, which retains only given components of tuples. For instance, the projection of x = <e1, e2, e3, e4> onto its first and third components is <e1, e3>.

More formally, let an element x of E^* be defined given an integer L_x and a function :

$x : (1 \,..\, L_x) \longrightarrow E$

so that x is denoted by :

$x = \prod_{i \leq L_x} x(i)$.

For a subset J of N, the projection onto J is defined as the function :

$$Pr_J : E^* \longrightarrow E^*$$
$$x \longmapsto \prod_{i \in J \cap (1 \,..\, L_x)} x(i).$$

These projections commute with any monoide homomorphism, and thus with substitutions. They define new invariant types closely related to the identity one except that some components are left. Moreover, composing such projection with the class function cl provides other invariant types.

5. The size.

The last invariant type we give is provided by the size function :

$$|\ |\ : Z(E_n^* \cup V^*) \longrightarrow Z.$$

Extending any substitution on Z by the identity, $S(|m|) = |S(m)|$ holds for any varible tuple multiset, and we have:

$$(M \gt M' \Longrightarrow w.|M_d| = w.|M'_d|)$$
$$\text{iff } w.|C| = 0.$$

This invariant type is very important, since it is the only one concerning ordinary Petri Nets. In fact, such an invariant is an invariant of the underlying net (cf definition 5).

Composing the size with function D defined above yields another invariant type.

6. Data invariants.

We call data invariants those whose computation involves not only the incidence matrix but also the value and structure of marks.

They can't by defined as the same for all High Level Petri Nets since they depend on the Data Model used to define marks. Moreover, looking for data invariant of a net is difficult to carry in a systematic way from general invariant types since such an invariant is directely related to the net data structure. For these reasons, we don't give results like definition 11 and theorem 4 for data invariants but only an example.

Let consider the seller and customer net represented on figure 1; we will proof an invariant assertion which means :

The price of merchandises which come out the Stock ends up in the Cashbox.

Let p be a Property of an Entity Class c, D be the range of value of p and E an Entity set. We define a function giving the value of E Entities on p as follows:

Val_p : E ----> D U 0
e :---> e.p if the class of e is c,
0 else.

Now we define the function $V = Val_{amount} + Val_{price}$:

V : E ----> R
e :---> e.amount if e is a Payment,
e.price if e is a Merchandise,
0 else

Finally we are interested in the function $F = V \circ Pr_1$:

F : E^* ----------> R
<el, . ., en> :---> V(el)

and our invariant is :

$W = (w_p.F)$, where $w = (w_p) = (1, 0, 0, 0, 1, 1, 0, 1, 0, 0)$. Thus W is the function F on the places Stock, p_5, p_6 and Cashbox, and is null on the others according to the above given meaning.

Let M and M' be two markings of the net such that there exists a loaded transition (t, S) with M ((t, S)> M'. Thus $M'_d - M_d = S(C).\underline{t}$ and we ask for $W.S(C).\underline{t} = 0$.

This obviously holds for every transition except for t_1 and t_5.

Concerning t_1, we have :

$W.S(C).\underline{t_1} = F(S(price)) - F(S(merchandise))$.

But according to t_1 Action

S(price).amount = S(merchandise).value

so $W.S(C).\underline{t_1} = 0$.

The same holds concerning t_5 where the variables price and merchandise have reversed parts.

Thus W is an invariant of the net and for two markings M and M' :

M (> M' ==> $(w_p).F(M_d) = (w_p).F(M'_d)$.

7) Conclusion

PNDS are a suitable tool in order to model the Information System working, as set in (SIB 84). They satisfy the requirements of a dynamics model in Office Automation, and they enable to process the following operations on information Systems :

- specification, design,
- reliable and precise descriptions,
- be a communication support,
- property analysis, validation,
- simulation.

An Information System may be viewed as a set of related Office Procedures (ZIS 75) which are described by normalized, without ubiquity and coherent PNDS. This is done by a mapping from concepts used to describe Office Procedures into those used to define a PNDS, so that we have a rigorous translation from Office Procedures into PNDS, and conversely an interpretation of PNDS in the Information System area.

The same way is followed for Information System property analysis : a correspondence between the Information System properties and the PNDS properties is defined, so that an Office Procedure is studied by studying its PNDS.

PNDS formalism is not yet integrated into Analysis Methodology as dynamics model, but this will be undertaken.

BIBLIOGRAPHIE

(BRAMS) BRAMS G. W.
Réseaux de Petri : théorie et pratique
Masson, 1983 Paris, 2 tomes

(FER83) FERBER J.
MERING : un langage d'acteurs pour la représentation et la manipulation des connaissances
these Docteur Ingenieur, Universite Paris 6, dec 83

(GENR 81) GENRICH H. J., LAUTENBACH K.
S-invariance in Predicate/transitions nets
Application and theorie of Petri nets, IF 66, Springer 83

(JENSEN 82) JENSEN K.
Coloured Petri nets and the invariant method
Application and theorie of Petri nets, IF 66, Springer 83

(LAUT74) LAUTENBACH K., SCHMID H.
use of Petri Nets for proving correctness of concurrent Process System
Information processing 74, North-Holland 74

(MINSKY) MINSKY M.
A framework for representing knowledge
The psychologie of computer vision, P. H WINSTON éditeur, Mc-Graw Hill 75

(OBER) OBERQUELLE H.
Some concepts for studying flow and modification of actors and objects in High-Level Petri Nets
3 European Workshop on applications and theory of Petri Nets, Varenna, sept 82

(REIS) REIZIG W.
Petri nets with individual tokens
Application and theorie of Petri nets, IF 66, Springer 83

(HOA 69) HOARE C.A.R.
An axiomatic basis for computer programming
Communication ACM 12,10, oct 1969

(RICH) RICHTER G.
IML-inscribed Nets for modeling text processing and Data (base) Management Systems
7th conf. on VLDB, Canne 81

(SIB84) SIBERTIN-BLANC C.
La modelisation des Procedures Administratives par des Reseaux de Petri a Structure de Donnees
These de Docteur Ingenieur, Toulouse, dec 84

(VAUTH) VAUTHERIN J., MEMMI G.
Computation of flows for unary-predicates/transitions-nets
5th European Workshop on Applications and Theorie of Petri Nets, juin 84, Aarhus, Danemark

(ZIS 75) ZISMAN M.
A system for Computerization Of Office Procedures
PhD dissertation, Warton School, 75